BASIC USER'S GUIDE

IF YOU DON'T KNOW BASIC,
YOU WON'T LEARN IT HERE

by Michael Gallo

With no code examples
WHATSOEVER

Table of Contents

What is 8ASIC....3
Conventions....3
Hotkeys....3
8ASIC vs. 8-bit BASICs....4
Saving and Loading....5
Speed and Interrupts....6
Variables, Strings and Arrays....7
Color....8
Vector Graphics....9
Scrolling....10
Sprites....11
Sound....12
Speech....13
Keyboard and Mouse....14
BASIC Keywords....15
Built-in Functions....30
Operators....35
System Variables....36
Special Characters....45
Included Demos....46
Open Source Licenses....49

What is 8ASIC

8ASIC is a BASIC language interpreter, loosely based on BASIC interpreters of the 8-bit era, like those found on the ATARI 800, the Commodore C64, or the Sinclair ZX Spectrum. It was designed to run old-school BASIC programs, without being an emulator of any particular machine. It pays homage to the ATARI 800, because that was my first computer, but it's by no means limited to that particular BASIC dialect. On the contrary, 8ASIC implements a super-set of keywords and features, originating from multiple machines of the 8-bit era.

Conventions

This guide uses the following conventions:

- *aexpr* means an arithmetic expression
- *bexpr* means a Boolean expression
- *sexpr* means a string expression
- *expr* means any kind of expression
- [square brackets] denote optional arguments or keywords
- `8ASIC` font indicates commands and code fragments
- BASIC keywords are written in UPPERCASE, even though they are case-insensitive

Hotkeys

Key	Function	Key	Function
F1	Open this guide	Cmd/Ctrl+B Ctrl+Pause	Interrupt execution
F5	Warm reset	Cmd/Ctrl+O Insert	Toggle overwrite mode
Shift+F5	Cold reset	Cmd/Ctrl+P Pause	Pause execution
F11 Alt+Enter	Toggle fullscreen		

8ASIC vs. 8-bit BASICs

While 8ASIC is based on 8-bit era BASIC interpreters, there are a few significant differences you need to be aware of, if you're used to programming on vintage machines.

- Lines with lower line numbers are not executed any faster than those with higher numbers.
- Functionality which would traditionally be accessed via **PEEK** and **POKE** has been exposed via system variables.
- There's no ATARI-style command abbreviation, although some common abbreviated forms of commands, like **?** and **PR** for **PRINT**, **.** for **REM**, and **#** for **LABEL** are supported.
- No I/O commands for reading from/writing to floppy or cassette. Instead, 8ASIC uses your computer's normal file system.
- Strings can be dimensioned, but don't have to be. In either case, they are fully dynamic, and can grow to any size. Arrays (both numerical and string) have to be dimensioned.
- The parsing of **DATA** entries was incompatible between different vintage machines. I've chosen the C64 format, with its quoted string support, and trimming of leading and trailing spaces.
- **CONT** will continue execution from the next statement, not the next line.
- The screen editor is in insert mode by default, like modern text editors. To switch to overwrite mode, press *Insert* or *Cmd/Ctrl+O*.
- Color and graphics capabilities of vintage machines varied quite a bit. 8ASIC uses a modern color representation, it features high-resolution vector graphics, and a powerful sprite system.
- Likewise, 8-bit computers had different sound chips (sometimes even none at all). 8ASIC has a 16-voice software synthesizer, and can utilize your OS's text-to-speech functionality.
- Although 8ASIC's font contains many special characters from the ATASCII and PETSCII character tables, as well as other characters that are unique to 8ASIC, it's currently not possible to replace them with custom characters.

Saving and Loading

Because 8ASIC is not an emulator, it uses your computer's regular file system to save and load programs.

LOAD "Hello" will try to load the file "Hello.bas" from the 8ASIC sub-folder in your OS's default documents folder. You can also specify the entire path, and/or file extension as well.

If you don't recall the name or location of the program you want to load, simply enter **LOAD** without a file name. 8ASIC will open an OS file dialog, allowing you to find the desired file interactively.

The **SAVE** command works pretty much the same way, except in the opposite direction. You can also merge a previously saved program with the currently loaded one, using the **ENTER** command. Just look out for potential line number conflicts, because any newly-loaded lines will replace existing ones with identical line numbers.

Finally, there's one more command, which makes it more convenient to load the demo programs included with 8ASIC. It's called **DEMO**, and it works more or less the same as the **LOAD** command, except it loads programs from 8BASIC's internal demo folder by default.

Besides BASIC programs, 8ASIC can also load JPEG and PNG files to be used as sprites using the **SPRITEMAP** command.

Also, the output of 8ASIC's software synthesizer can be saved to a WAV file using the **RECORD** command.

File I/O from within BASIC programs is currently not available, but might be added in a future release.

Speed and Interrupts

8ASIC has two execution modes: fast and slow, while the speed of the latter is adjustable. Fast execution mode is enabled by issuing the **FAST** command, and in this mode, 8ASIC will execute code as fast as it can.

Slow mode is in turn activated by the **SLOW** command. Without any arguments, **SLOW** sets the execution speed to the default rate of about 200 lines per second. This is also the speed to which the virtual machine is set initially. It enables the common BASIC practice of introducing delays into the program by running an empty **FOR** loop for a number of iterations. To set a specific speed in lines per second, simply pass it as an argument to the command. The maximum speed in **SLOW** mode is 1000 lines per second, since the granularity of the delay is 1 millisecond.

If your program is writing text or drawing graphics on the screen, its speed might also be limited by the automatic refresh after each screen-altering command. The **REFRESH** command can be used to suspend automatic screen refresh, and to achieve the highest possible performance.

Program execution can be paused at any time by pressing the *Pause* key, or *Cmd/Ctrl+P*, and stopped by pressing the *Break* key, or *Cmd/Ctrl+B*. If execution was stopped in this way, it can be resumed using the **CONT** command. Another way to stop program execution is to “warm reset” the virtual machine by pressing *F5*. This also resets most system variables to their default values, stops all sound coming from the software synthesizer, and also stops any sound recording in progress. To perform a “cold reset”, which also erases the program and all variables, press *Shift+F5*.

Program execution can also be interrupted in a timed manner using the **AFTER** and **EVERY** commands. The former creates a one-shot interrupt, while the latter establishes a periodic one.

When an interrupt is triggered, 8ASIC executes a jump to subroutine (**GOSUB**) to the specified line. Normal execution can then be resumed using the regular **RETURN** command. Note that if the interrupt function takes too much CPU time, it's possible that the next interrupt will trigger immediately after the **RETURN** command, and there won't be any time for normal program execution.

Interrupts can be suppressed using the **INTERRUPTS** command, and their status can be queried by reading the **@INTERRUPTS** system variable.

Variables, Strings and Arrays

Variable names have to start with a letter, and they can contain any number of letters, numbers, and underscores. They are case-sensitive, unlike keywords. String variable names have to end with a dollar sign ($).

Like with most BASICs, all numerical variables are capable of storing double-precision floating-point numbers. 8ASIC's variables can also store 64-bit signed integer numbers. Strings can be of any length, and they contain UTF32 characters. Of course, 8ASIC's custom font does not have glyphs for all possible UNICODE characters, but most European languages are covered, as are the ATASCII and PETSCII code tables (to the extent where those characters exist within the UNICODE standard).

In subroutines, (entered via **GOSUB**) you can declare local variables using the **LOCAL** keyword. These will take precedence over any existing global or local variables with the same name, and will cease to exist after the next **RETURN**. They can be especially useful for recursive algorithms. **FOR** loop counters are always local.

Global variables can be listed using the **GLOBALS** command, and if the program is interrupted in a subroutine, you can list local variables as well, using the **LOCALS** command.

Arrays can have any number of dimensions, and any size, (within reason) provided they are correctly dimensioned first (see **DIM**). String arrays, even multi-dimensional ones, are supported as well. Non-array strings can, but don't have to be dimensioned. When dimensioning string arrays, don't forget to add one more dimension (actual value doesn't matter). This is for compatibility reasons.

Both ATARI/Commodore style and Sinclair style sub-strings are supported; **A$(5,7)** and **A$(5 TO 7)** will both work. Sub-strings can also be used on both sides of the equals sign (as L-values and R-values). To specify a sub--string of a string array item, simply do so within the same set of brackets. **R$(5,3 TO 6)** will return the 3rd to 6th characters of the 5th item of the string array **R$**.

Unlike most 8-bit BASICs, 8ASIC features string concatenation using the + operator.

Some basics had no possibility to enter a double quotation mark (") into a string literal. 8ASIC supports the two double quotation marks ("") escape sequence.

Color

8ASIC uses a 32-bit color representation, with 8 bits reserved for opacity, and 8 bits for each of the red, green, and blue color components. Opacity, or alpha, is stored in the highest byte.

Vintage BASICs often didn't understand hexadecimal numbers, but 8ASIC does, and it's easy to express colors that way. A half-transparent, bright green color for example, would be written as $7F00FF00. Using bitwise operators, and some careful bit-shifting, one could split this number into the individual color components, and then put it back together again. However, there's another, more convenient way.

The built-in **RGB()** and **RGBA()** functions take the individual components, and combine them into a single number, sparing you the effort of doing so with bit shifts and bitwise-ORs. In this example, RGB stands for red, green, and blue, and the function always creates opaque colors. RGBA means red, green, blue, and alpha, where alpha is opacity.

Splitting colors into their components is also covered, in the form of built-in **RED()**, **GREEN()**, **BLUE()** and **ALPHA()** functions.

Color-related commands varied quite a bit from one vintage machine to the next. 8ASIC has a **PEN** command, which sets the foreground color of text and graphics alike, and a **BRUSH** command, for text background and filled shapes. The overall background color of the screen is set using the **PAPER** command.

If **PEN** and **BRUSH** are used in-line within a **PRINT** statement, they set the pen or brush color for the following characters, until the end of the statement.

8ASIC also has a built-in palette of named colors, which can be accessed using the **COLOR()** and **COLOR$()** functions, and the **@COLORS** system variable.

@COLORS holds the total number of colors in the palette.

COLOR() can be used with a numerical argument, in which case it's interpreted as a one-based index into the alphabetically-sorted palette. If the argument is negative, it's instead interpreted as an index into the palette sorted by hue and lightness. If a string argument is provided, it's matched against all the color names, and if a match is found, the function returns that color. In case of no match, the function returns 0.

Finally, **COLOR$()** returns the name of the color in the argument, if it exists in the palette.

String literals or variables holding color names can also be directly used as arguments for the **BRUSH**, **PEN** and **PAPER** commands.

Vector Graphics

Unlike emulators, 8ASIC doesn't attempt to re-create the quirks of graphics chips, and low-resolution CRT displays of vintage machines. Instead, it implements an idealized version of BASIC graphics commands, utilizing the full resolution of today's screens.

The range and origin of graphics coordinates are adjustable, and default to 8 times text resolution, and bottom left corner of the screen respectively. They are controlled by the **GRAPHICS** command, and the **@GRAPHX** and **@GRAPHY** system variables.

GRAPHICS has two arguments (for X and Y), and it sets the range of the graphics coordinates relative to the text resolution. For example, **GRAPHICS 8,8** would set the default range of 8 times text resolution. If only one argument is provided, the same value will be used for both X and Y coordinates. Note that the multiplier(s) don't have to be integers.

The **@GRAPHX** and **@GRAPHY** system variables can be used to query the current absolute range of graphics coordinates, but it can also be used to set it. In that case, the text-to-graphics multiplier will be calculated from the absolute value(s). In both cases, if the text resolution changes for any reason, so will the range of graphics coordinates.

To move the origin to the top of the screen, or even to the right side, simply negate one or both of the arguments passed to the **GRAPHICS** command. Similarly, if setting the range via the **@GRAPHX/Y** system variables, negate one or both values to move the origin to the top and/or right.

Besides the standard points and lines, 8ASIC can also draw boxes, (**BOX** command) circles and ellipses, (**CIRCLE** command) and arbitrary polygons (**VERTEX** command). These shapes are always outlined using the current pen, and filled using the current brush.

Turtle graphics is also available, via the **PENDOWN/UP**, **BRUSHDOW/UP**, **LEFT**, **RIGHT**, **FORWARD**, **BACKWARD**, and **HEADING** commands, and the **@TURTLE**, **@HEADING**, **@PENDOWN** and **@BRUSHDOWN** system variables.

Graphics can be drawn in front of, or behind text. This is controlled by the **FOREGROUND** and **BACKGROUND** commands. Each of the three screen layers (background, text, and foreground) can be independently cleared, without disturbing the contents of the other two layers. This is done by passing an argument to the **CLS** command, with the value of 1 for the background, 2 for text or 4 for the foreground. These values can also be added together, to clear multiple layers at once. Note that re-ordering the layers using the **@TEXTLAYER** system variable doesn't change the meaning of these values.

Scrolling

The most basic way to achieve scrolling in 8ASIC is to simply print more text than fits on the screen, which will cause it to scroll up automatically. This kind of “coarse” text scrolling, by whole lines or columns, can also be performed at any time, and in any direction, using the **SCROLL** command.

The **@NUDGE** system array can be used to offset each of the three screen layers (background, text, and foreground) by any distance, even smaller than one unit in graphics coordinates. If used in combination with the **SCROLL** command, it enables smooth (or “fine”) text layer scrolling.

To hide the process of adding new text lines or columns at the edge of the screen, set the **@OVERSCAN** system variable to a non-zero value. This will extend all screen layers by two characters in every direction, ensuring that at least one row or column of characters is always beyond the screen edge. This is where you can add new characters invisibly.

Scrolling the graphics layers is also achieved using the **@NUDGE** system array. Because there are two layers, you can scroll by one complete screen before having to draw more graphics. To place both layers in front of, or behind text, change the **@TEXTLAYER** system variable.

Large scrolling graphics can also be achieved with just one layer, by switching it to vector mode using the **VECTOR** command. In this mode, graphics drawn on that layer are not rasterized immediately, but are kept as vectors, and rasterized during each screen refresh. This slows down screen updates, especially with very complex graphics, but it allows you to draw images that are much larger than the screen, and then **@NUDGE** them around to reveal different parts.

Sprites

Besides the main CPU, some vintage machines had special chips, that could draw additional objects over (or under) the normal "playfield" graphics. These objects were commonly called sprites, although ATARI preferred the terminology "players" and "missiles".

8ASIC supports loading single sprites, or entire sprite maps (also known as "sprite sheets") from JPEG or PNG images, using the **SPRITEMAP** command. In case of PNG, transparency is fully supported. On sprite maps, multiple sprites or multiple animation frames have to have the same size, and have to be arranged in rows and columns.

Once a sprite map is loaded, sprites can be set up to display frames from it using the **SPRITEDEF** command, with either trilinear or nearest neighbor sampling. Sprites can be freely positioned, rotated, and scaled (proportionally or stretched) using the **SPRITEPOS** command. The can also be placed in front or behind any of the three screen layers, made semi-transparent, or set to use an alternate blending mode using the **SPRITEBLEND** command.

Sprite parameters can also be accessed via system arrays. **@MAP** controls from which sprite map the sprite will source its content, while **@FRAME** sets which frame from the sprite map will be displayed. **@OPACITY** sets each sprite's overall opacity, **@BLENDMODE** sets the blending mode, **@FILTER** turns the filtering on or off, and **@LAYER** specifies where in the screen drawing order the sprite will be drawn. The position of each sprite is controlled by **@POSX** and **@POSY** arrays, rotation by **@ROTATION**, and uniform sprite scale by **@SCALE**. Sprites can also be scaled non-uniformly, by setting the corresponding items in the **@SCALEX** and **@SCALEY** arrays instead.

The number of active sprites is limited only by your machine's available video memory and GPU performance.

8ASIC currently lacks sprite collision detection, which in case of some vintage machines was implemented in the aforementioned special chips. This might be added in a future release, but fast collision detection of arbitrarily-transformed high-resolution bitmaps is not a trivial task.

Sound

In lieu of a sound chip, 8ASIC has a 16-voice software synthesizer, capable of generating multiple basic waveforms. Sounds and musical notes are triggered using the **SOUND** command, and the various properties of the 16 voices can be controlled via system variables.

The **SOUND** command can be issued with up to 4 arguments. Without any arguments, it acts as a "note off" to all channels. Depending on the release time of the channels' envelopes, (see **@RELEASE**) this might not turn off all sound immediately. To do that, reset the machine using the *F5* key.

The first argument of the **SOUND** command is the voice number, ranging from 1 to 16. If no further argument is given, this is interpreted as a "note off" for that particular voice.

The second argument is either a frequency in Hz, or a string containing a note name in scientific, or Helmholtz notation. If this second argument is provided, **SOUND** acts as a "note on" for the selected voice. The voice will start playing a sound of the specified frequency, or the frequency corresponding to the specified musical note, adjusted by the voice's entry in the **@PITCH** system array. The pitch of each voice can also vary over time, depending on the corresponding value in the **@PORTAMENTO** system array.

The third optional argument is the relative volume at which the sound will be played. The default volume, if the argument is not provided, is 0.75.

The fourth argument is the duration of the sound in seconds. If not provided or zero, the sound will play until a "note off" is issued for that same voice.

The shape of the waveform produced by each voice is determined by the corresponding values in the **@CLIP**, **@PULSE**, **@SLOPE**, **@SINE**, and **@NOISE** system arrays. The sound of each voice then goes through a low-pass filter, controlled by the **@LOWPASS** system array. Consult the System Variables section for the specifics.

Volume envelopes are based on the common *ADSR* paradigm, and they are controlled by each voice's entries in the **@ATTACK**, **@DECAY**, **@SUSTAIN**, and **@RELEASE** system arrays. In terms of mixing, each channel has a **@VOLUME** and **@PAN** parameter, with the latter employing the simple -3 dB pan rule.

When it comes to effects, there's **@TREMOLO** and **@VIBRATO**, and the frequency shared by both of them can be set by writing to the **@LFO** system array.

The output of the synthesizer can be recorded to a .WAV file using the **RECORD** command. It works in a way similar to the **SAVE** command, where a file name can be specified directly in an argument, or via an OS file save dialog. The recording can be stopped by resetting the virtual machine using the *F5* key.

Speech

Besides sound synthesis, 8ASIC also gives you access to the text-to-speech features of your operating system, via the **SAY** and **VOICE** commands, and **@VOICE**, **@VOICES**, **@VOICENAME$**, and **@VOICELOCALE$** system variables.

SAY works in a manner similar to the **PRINT** command, but instead of printing text on the screen, it reads it out aloud. Also similar to **PRINT**, separating expressions by a comma produces a slight pause in the speech, whereas a semicolon connects two expressions into a single utterance. Ending an utterance with a comma or semicolon causes the **SAY** command to block, until the utterance is spoken in its entirety. Otherwise, execution will continue immediately, and the utterance will be spoken in parallel.

Note that despite this optionally parallel nature of the **SAY** command, only one utterance can be spoken at a time. **SAY**ing a new one interrupts the one currently being said. Also, the output of **SAY** is neither recorded into the file saved by **RECORD**, nor is it interrupted by **END**, or by resetting the virtual machine using the *F5* key.

The names of the text-to-speech voices available to 8ASIC can be looked up in the **@VOICENAME$** array, and their total number in the **@VOICES** variable. The current voice is in **@VOICE**. For correct pronunciation, it's important to match the language of the voice, to the language of the text being spoken. The languages of the available voices can be looked up in **@VOICELOCALE$**.

To select a voice, you can use the **VOICE** command, either with a numerical argument, in which case it's interpreted as a one-based index into the voice array, or with a string argument, which is matched against all the available voice names. As an alternative, you can also set the **@VOICE** variable directly.

Keyboard and Mouse

Keyboard input in 8ASIC is mostly period-correct, with **GET**, **INKEY** and **INKEY$** working as you'd expect. In addition to these commands, you can also query the **@KEY** system array, to determine which keys are currently pressed. The key codes returned by **GET** and **INEKY** correspond to items in this array. Some of these codes are straight-forward ASCII values – 65 for A, 66 for B, and so on, others less so.

In immediate mode, the mouse can be used to move the text cursor. In run mode, the mouse position (in graphics coordinates) can be queried by reading the **@MOUSEX** and **@MOUSEY** system arrays. Note that because the 8ASIC window can't always be divided into a whole number of text characters in both X and Y directions, one of the mouse coordinates can be negative (if the mouse is at the very edge of the window).

The state of (up to 32) mouse buttons is stored in **@BUTTON**. The following table lists which item in this array corresponds to which button:

Index	Button
1	Left
2	Right
3	Middle
4	Back
5	Forward
6	Task
7-32	Any extra buttons your mouse might have

You can track mouse button presses and double-clicks via the **@CLICK** and **@DBLCLICK** system arrays. These are 2-dimensional arrays, where the first dimension is the button index, (see table above) and the second dimension is the coordinate, X (1) or Y (2), where that button was last pressed or double-clicked.

In order to track multiple clicks in the same location, you can set items in these arrays to impossible coordinates (-1000 for example), then wait until they change back to normal values. The same can be done with the **@MOUSEX** and **@MOUSEY** variables to track mouse movements.

BASIC Keywords

8ASIC implements a super-set of BASIC keywords from most 8-bit machines. The goal was to be able to type in almost any old BASIC program, and have it work with minimal or no changes. In cases where there was a conflict between the various BASIC implementations, I tried to pick the one which seemed most logical to me.

```
AFTER aexpr[,aexpr] GOSUB aexpr
AFTER aexpr[,aexpr] GO SUB aexpr
```

Sets up a one-shot timed interrupt that will occur after the period of time (in seconds) specified in the first argument. If the second, optional argument is also provided, it's interpreted as the interrupt's priority (with the default priority being 1). Note that two interrupts with the same priority can't be set, and setting the second one will replace the first one. The theoretical time resolution is one millisecond, but this can rarely be achieved in practice. To return from interrupt code, use **RETURN** like with any normal subroutine. See also **EVERY**.

```
AT aexpr,aexpr
LOCATE aexpr,aexpr
POSITION aexpr,aexpr
```

Moves the text cursor to the specified position. The arguments are interpreted as X and Y coordinates, with their origin in the top left corner of the screen.

```
BACKGROUND
```

After this command is executed, all subsequent vector graphics objects will be drawn in the background (behind text by default). This is the default. See also **FOREGROUND**, and **@TEXTLAYER**.

```
BACKWARD
```

Moves the turtle backward by the specified distance (in graphics coordinate units). See also **FORWARD**.

```
BOX aexpr,aexpr,aexpr,aexpr[,expr[,expr]]
```

Draws a box filled with the current brush color, and outlined with the current pen color. The first four mandatory arguments are interpreted as *x, y, width,* and *height*. The fifth and sixth arguments, if provided, are interpreted as pen and brush colors. If they are strings, these will be looked up in the palette of named colors.

BRUSH expr

Sets the background color for any subsequently written text characters, and fill color for boxes, circles, and polygons. Default is 0, meaning transparent. If the argument is a string expression, it will be looked up in the palette of named colors. **BRUSH** can also be used in-line within a **PRINT** statement, in which case it sets the brush color until the end of that statement. See also **PEN** and **PAPER**.

BRUSHDOWN

Puts the turtle's brush down (starts drawing a filled polygon behind the turtle). See also **BRUSHUP**, **PENDOWN**, **PENUP**.

BRUSHUP

Lifts the turtle's brush up (stops drawing a filled polygon behind the turtle). See also **BRUSHDOWN**, **PENDOWN**, **PENUP**.

BYE

Shuts down 8ASIC.

CIRCLE aexpr,aexpr,aexpr[,aexpr[,expr[,expr]]]

Draws a circle with the specified center (first two arguments, interpreted as X and Y coordinates), and radius (third argument). If an fourth argument is given, it's interpreted as the vertical radius, and if the horizontal and vertical radii don't match, the command draws an ellipse. The fifth and sixth arguments, if provided, are interpreted as pen and brush colors. If they are strings, these will be looked up in the palette of named colors.

CLR
CLEAR

Sets all variables to zero, frees all strings and arrays.

CLS [aexpr]

Clears the screen, and erases all vector graphics objects. If the optional argument is provided, it's interpreted as a bit mask, indicating which layers should be cleared. If bit 1 is set, the background will be cleared, bit 2 clears text, and bit 3 the foreground. **CLS 5** for example, will clear all graphics, both background and foreground, but not text.

CONT
RESUME

Continues execution from where it was stopped by an error or by pressing the *Break* or *Ctrl+B* key.

CURSOR bexpr

Turns the text cursor on or off, depending on the value of the provided Boolean expression. See also **@CURSOR**.

DATA item[,item2,...]

Creates data items, which can then be read into variables using the **READ** command. Data items can be strings or numbers, and are separated by commas. C64-style quoted strings are supported, and whitespace surrounding unquoted strings will be removed.

DEF [FN] name(variable[,variable,...])[=]expr

Defines, or re-defines a user function. Variables listed within the parentheses become the parameters of the function, and are local to it. The **FN** keyword is optional, as is the equals sign after the parameter list.

DEG

Switches the angular unit to degrees. See also **RAD**.

DEMO [sexpr]

Same as **LOAD** but defaults to the 8ASIC built-in demo folder, instead of the user's documents folder.

DIM array(aexpr[,aexpr,...])[,...]

Dimensions numerical or string arrays. Non-array string dimensioning is optional.

DRAW [TO] aexpr,aexpr[,expr,[expr]]
DRAWR aexpr,aexpr[,expr[,expr]]
DRAWTO aexpr,aexpr[,expr[,expr]]

Draws a line from the last coordinates to the ones passed to this command. In the case of **DRAWR**, the coordinates are interpreted as relative to the last ones used for any drawing command. The third and fourth arguments, if provided, are interpreted as the pen and brush colors. If they are strings, they will be looked up in the palette of named colors.

If the turtle's brush is down, these commands behave like the equivalent **VERTEX** or **VERTEXR** commands, in that they add a new point to the current filled polygon.

See also **PLOT**, **PLOTR**, **MOVE**, **MOVER**, **VERTEX**, and **VERTEXR**.

END

Ends any currently running program, clears the return stack, stops all timed interrupts, and sends a "note off" command to all synthesizer voices.

ENTER sexpr

Same as **LOAD**, but merges the program from file with the one already in memory. In case of line number conflicts, the newly loaded lines take precedence.

EVERY aexpr[,aexpr] GOSUB aexpr
EVERY aexpr[,aexpr] GO SUB aexpr

Sets up a periodic timed interrupt that will occur each time the period of time (in seconds) specified in the first argument elapses. If the second, optional argument is also provided, it's interpreted as the interrupt's priority (with the default priority being 1). Note that two interrupts with the same priority can't be set, and setting the second one will replace the first one. The theoretical time resolution is one millisecond, but this can rarely be achieved in practice. To return from interrupt code, use **RETURN** like with any normal subroutine. See also **AFTER**.

FAST

Instructs 8ASIC to execute code as fast as possible. On current machines, this means hundreds of thousands of lines per second. Note that the speed of commands that output to the screen may also be limited by the screen refresh frequency, depending on the **REFRESH** mode. See also **SLOW**.

[FN] name(expr[,expr,...])

Call a previously defined user function. The **FN** keyword, like **LET** is optional.

FOR counter=aexpr TO aexpr [STEP aexpr]

Opening statement of a **FOR** loop, which needs to be matched with a corresponding closing **NEXT** statement. For compatibility reasons, 8ASIC replicates the ATARI bug where each loop is executed at least once, even if the end of the range is beyond the start.

FOREGROUND

After this command is executed, all subsequent vector graphics objects will be drawn in the foreground (in front of text by default). See also **BACKGROUND**, and **@TEXTLAYER**.

FORWARD

Moves the turtle forward by the specified distance (in graphics coordinate units). See also **BACKWARD**.

GET variable

Retrieves a keyboard code or character from the keyboard buffer. The buffer records the last 10 keypresses. If *variable* is scalar, it will be set to the keyboard code of the first key in the buffer, or to zero if no keys were pressed. If *variable* is a string, it will be set to the character generated by the first key in the buffer, or to an empty string if none. See also **INKEY**, **INKEY$**, and **@KEY**.

GLOBALS

Lists all existing global variables and their values.

GOSUB aexpr
GO SUB aexpr

Pushes the current program location on the stack, then jumps to specified line number. Note that like on the ATARI, the line number can be any numerical expression.

GOTO aexpr
GO TO aexpr

Jumps to the specified line number. Same as with **GOSUB**, the argument can be any numerical expression.

GRAPHICS aexpr[,aexpr]

Sets both **@GRAPHX** and **@GRAPHY** system variables at once, relative to text resolution. For example, **GRAPHICS 8,8** sets the coordinate range to the default 8 x text resolution in both X and Y directions. If only one argument is provided, it will be used for both X and Y coordinates.

HEADING aexpr

Sets the turtle's heading absolutely. The current heading can be queried using the **@HEADING** system variable. See also **LEFT**, **RIGHT**.

HELP

Opens this guide. It can also be opened by pressing F1.

IF bexpr THEN ...

Evaluates the expression between **IF** and **THEN**, and if the result is non-zero, continues execution after **THEN**. If the result is zero, the execution continues from the next line. If the statement following **THEN** is an arithmetic expression, it is evaluated, and interpreted as a line number to jump to if *bexpr* is non-zero. If the statement following *bexpr* is a **GO...** command, the **THEN** token can be omitted.

INKEY

Returns the keyboard code of any currently pressed key. If multiple keys are pressed, returns the code of the last one. See also **GET**, **INKEY$**, and **@KEY**.

INKEY$

Returns the character corresponding to any currently pressed key. If multiple keys are pressed, returns the character corresponding to the last one. See also **GET**, **INKEY**, and **@KEY**.

INPUT [expr;]variable[,variable,...]

If **INPUT** is followed by an expression and a semicolon, the result of that expression will be printed on the screen, followed by a question mark. Otherwise, just a question mark will be printed. 8ASIC will then wait for the user to enter comma-separated strings or numbers, which will then be stored in the specified variables. String variables accept any input, numerical variables accept only numbers. If the user inputs fewer than the requested number of items, **??** will be printed, allowing the user to continue entering more data, until all variables passed to **INPUT** are filled.

INTERRUPTS bexpr

Turns interrupts on or off, depending on the value of the Boolean expression. Note that interrupts work only while the program is running, regardless of this flag. See also **@INTERRUPTS**.

# variable
LABEL variable

Creates a read-only variable set to the line number of the line on which this command is located. This variable can then be used as a target for **GOTO** / **GOSUB** commands, but also in any other numerical expression.

LEFT aexpr

Turns the turtle to the left by the specified angle. See also **HEADING**, **RIGHT**.

[LET] variable=aexpr
[LET] variable$=sexpr

Optional keyword indicating variable assignment.

LIST [aexpr[,aexpr]]

Lists the program currently residing in memory. The optional first argument can be used to start the listing from a particular line. If two arguments are provided, they are interpreted as a range of lines to be listed. Either of the two arguments can be omitted, indicating listing from the start / to the end of the program.

LOAD [sexpr]

Loads a program from the specified file into memory. If the path is relative, 8ASIC will look for the file in the "8ASIC" sub-folder of the user documents folder. Adding a .bas file extension is also optional, as it will be added automatically. If no argument is given, **LOAD** opens an OS file dialog, permitting you to select the file interactively.

LOCAL variable[,variable,...]

Declares one or more local variables. These variables override any global variables, and are only valid in the current scope (until **RETURN**). The command has no effect if the return stack is empty (no **GOSUB** command was executed). Note that all **FOR** loop counters are automatically local.

LOCALS

Lists all local variables. This only works if used within a subroutine, or if the program was interrupted while a subroutine was running.

```
MOVE  aexpr,aexpr[,expr[,expr]]
MOVER aexpr,aexpr[,expr[,expr]]
```

Moves the origin point of the next **DRAW** command to the specified coordinates, without drawing a dot there, like **PLOT** would do. In the case of **MOVER**, the coordinates are interpreted as relative to the last ones used for any drawing command.

Note that if the turtle's pen is down, the behavior of this command changes to that of a **PLOT** or **PLOTR**. If the turtle's brush is down, this command instead behaves like **VERTEX** or **VERTEXR**.

The third and fourth arguments, if provided, are interpreted as pen and brush colors. If they are strings, these will be looked up in the palette of named colors. The colors are obviously only relevant if the turtle's pen and/or brush are down.

See also **DRAW**, **DRAWR**, **PLOT**, **PLOTR**, **VERTEX**, and **VERTEXR**.

```
NEXT  [counter]
```

Closing statement of a **FOR...NEXT** loop. The counter variable, if specified, needs to match the one used in the opening **FOR** statement.

```
NEW
```

Erases the current program, and all variables, strings, and arrays.

```
OLD
```

Restores a program erased by **NEW** or a cold reset, or overwritten by **OLD**, **LOAD**, or **DEMO**.

```
ON  aexpr  GOTO  aexpr[,aexpr,...]
ON  aexpr  GOSUB  aexpr[,aexpr,...]
ON  aexpr  GO  TO  aexpr[,aexpr,...]
ON  aexpr  GO  SUB  aexpr[,aexpr,...]
```

Evaluates the expression between **ON** and **GO**..., and interprets its value as a one-based index into the comma-separated list of expressions following the **GO**... command. If the value is out of range, execution simply continue with the next command. If one of the expressions is successfully chosen, the **GO**... command is then executed with the expression's value as the destination line number.

```
PAPER  aexpr
```

Sets background color of the screen. This color is visible wherever there is nothing drawn or written. Default is a custom royal blue color. If the argument is a string expression, it will be looked up in the palette of named colors. See also **BRUSH** and **PEN**.

PEN expr

Sets the foreground color for any subsequently written text characters, and outline color for lines, boxes, circles, and polygons. Default is **$FFFFFFFF**, meaning opaque white. If the argument is a string expression, it will be looked up in the palette of named colors. **PEN** can also be used in-line within a **PRINT** statement, in which case it sets the pen color until the end of that statement. See also **BRUSH** and **PAPER**.

PENDOWN

Puts the turtle's pen down (starts drawing a line behind the turtle). See also **BRUSHDOWN**, **BRUSHUP**, **PENUP**.

PENUP

Lift the turtle's pen up (stops drawing a line behind the turtle). See also **BRUSHDOWN**, **BRUSHUP**, **PENDOWN**.

PI

3.1415926535897932384626433832795

PAUSE aexpr

Pauses execution for the specified number of seconds (can be fractional).

POP

Removes the topmost, most recent entry from the loop / return stack. This is necessary in order to be able to leave a **FOR** loop early (via **GOTO**).

PLOT aexpr,aexpr[,expr[,expr]]
PLOTR aexpr,aexpr[,expr[,expr]]

Draws a circular dot at the specified coordinates. In the case of **PLOTR**, the coordinates are interpreted as relative to the last ones used for any drawing command. The size of the dot is usually large enough so that two dots spaced 1 unit apart, at the default coordinate range, touch each other. This is however dependent on the OpenGL implementation in use on the host machine.

The third and fourth arguments, if provided, are interpreted as pen and brush colors. If they are strings, these will be looked up in the palette of named colors. The brush color is only relevant if **PLOT** or **PLOTR** is used to start a new filled polygon, or if the turtle's brush is down, in which case these commands behave like **VERTEX** and **VERTEXR**.

See also **DRAW**, **DRAWR**, **MOVE**, **MOVER**, **VERTEX**, and **VERTEXR**.

`? expr[;expr][,expr][...]`
`PR expr[;expr][,expr][...]`
`PRINT expr[;expr][,expr][...]`

Evaluates and displays expressions on the screen. The expressions can be of any type, and they can be separated by semicolons or commas. Expressions separated by a semicolon will be displayed without any intervening whitespace. A comma on the other hand will first move the cursor to the next multiple of 8 characters, before printing the following expression. Ending the list of expressions with a semicolon or a coma will prevent the cursor from automatically advancing to the next line.

PEN and **BRUSH** commands can be used within a **PRINT** statement, to set the pen and brush colors until the next such command, or the end of the **PRINT** statement. The special **TAB()** function can also be used within a **PRINT** statement, to set the cursor position.

See also **TEXT**.

`PUT aexpr`

The equivalent of **PRINT CHR$(aexpr);**.

`RAD`

Switches the angular unit to radians. This is the default. See also **DEG**.

`RANDOMIZE [aexpr]`

Sets the seed of the pseudo-random number generator to the specified value, or to the current time since epoch in milliseconds, if no argument is given.

`RASTER`

Sets the current graphics layer (background or foreground) into raster mode, which means that any graphics drawn onto this layer will be clipped to the screen area, and rasterized immediately. This is the default mode. See also **VECTOR**.

`READ variable[,variable...]`

Reads a one or more **DATA** item(s) into the provided variable(s). Any type of item can be read into a string variable, whereas only numbers can be read into numerical variables. See also **RESTORE**.

RECORD [sexpr]

Starts recording the output of the software synthesizer to the specified file. If the path is relative, 8ASIC will save the file in the "8ASIC" sub-folder of the user documents folder. Adding a .WAV file extension is also optional, as it will be added automatically. If no argument is given, **RECORD** opens an OS file dialog, permitting you to choose the output file interactively.

REFRESH [bexpr]

8ASIC uses implicit double-buffering. Normally, the buffers are flipped after every command which alters the contents of the screen, either graphical or text. This behavior can be controlled using the **REFRESH** command. If provided an argument, **REFRESH** turns automatic buffer flipping on or off, based on whether the expression is non-zero or zero. Without an argument, it triggers a buffer flip. This allows you to draw many objects on the screen invisibly, then reveal them all at once. Note that buffer flipping is always synchronized with the vertical blanking period of your display.

. text
REM text

Makes the parser ignore anything beyond this keyword, until the end of the program line.

RENAME variable,new

Renames *variable* to *new* throughout the entire program.

RENUMBER [aexpr[,aexpr[,aexpr[,aexpr]]]]

Changes the numbers of program lines according to the arguments given. The first argument is the new starting line number (10 if not provided). The second argument is the step by which the line number will be increased for each subsequent line (10 if not provided). The third and fourth arguments specify the first and last original line number between which renumbering should happen. They default to the first and last line respectively.

Note that automatic renumbering can't deal with **GOTO** and **GOSUB** line numbers that are calculated from numerical expressions. If such a destination line number is encountered, **RENUMBER** will print a warning.

RESTORE [aexpr]

If used without argument, it resets the read location to the first **DATA** item in the program. If an argument is provided, the read location is set to the first **DATA** item found on or after the line with that number.

RETURN

Pops the topmost return location from the stack, then jumps to it.

RIGHT aexpr

Turns the turtle to the right by the specified angle. See also **HEADING**, **LEFT**.

RUN [aexpr]

Runs the current program from the specified line, or from the start if no line is given. Also performs an implicit **CLR**.

SAVE [sexpr]

Saves the current program to the specified file. If the path is relative, 8ASIC will save the file in the "8ASIC" sub-folder of the user documents folder. Adding a .bas file extension is also optional, as it will be added automatically. If no argument is given, **SAVE** opens an OS file dialog, permitting you to choose the save location and file name interactively.

SAY expr[;expr][,expr][...]

Evaluates and reads out expressions using the text-to-speech engine built into your OS. The expressions can be of any type, and they can be separated by semicolons or commas. Expressions separated by a semicolon will be read out one after another, while a comma will introduce a brief pause. Ending the list of expressions with a semicolon or a coma, will cause 8ASIC to wait until the text-to-speech engine finishes reading the entire utterance. Otherwise, the execution will continue immediately. See also **VOICE**, **@VOICES**, **@VOICE**, and **@VOICENAME$**.

SCROLL [aexpr[,aexpr]]

If no argument is provided, this command scrolls the screen up by one line. One argument is interpreted as the number of lines to scroll up (positive) or down (negative). Two arguments are interpreted as X and Y scroll distance in characters.

SLOW [aexpr]

Slows down execution to a "period correct" speed. The default setting, if the parameter is omitted, is 200 lines per second. The maximum speed in slow mode is about 1000 lines per second, since the granularity of the delay is 1 ms.

SOUND [aexpr[,expr[,aexpr]]]

Without arguments, **SOUND** sends a “note off” to all software synthesizer voices. With one argument, it acts as a “note off” for that particular voice only. The second argument can either be a frequency in Hz, or a string containing a musical note in the scientific or Helmholz notation; **"C#4"**, **"Bb3"**, **"c'"**, **"F,,,"** and so on. In this case, **SOUND** sends a “note on” to the selected voice, with the specified frequency, or the frequency corresponding to the specified musical note. The octave range for musical notes is from 0 to 8, or sub-contra to 5-line. The third optional argument is the sound's relative volume as a fraction from 0 to 1. The default volume is 0.75. The fourth optional argument is the sound's duration in seconds. If non-zero, it will cause a “note off” to be sent to the same voice automatically, after the specified time period. This is especially useful for sound effects.

SPRITEBLEND aexpr,aexpr[,aexpr[,aexpr]]

Sets the opacity of the sprite with the number specified in the first argument to the value of the second argument, where 0 is transparent and 1 is opaque. The optional third argument can be used to set where in the screen drawing order the sprite will be drawn. Valid values are from 1 (behind all layers) to 4, (in front of all layers) which is also the default. The optional fourth argument can be used to set the sprite's blending mode, which can be either 1 for normal blending, or 2 for “screen” mode blending useful for some special effects.

See also **@OPACITY, @LAYER** and **@BLENDMODE**.

SPRITEDEF aexpr,aexpr[,aexpr[,bexpr]]

Sets the sprite with the number specified in the first argument to use the sprite map with the ID in the second argument. The optional third argument can be used to display a particular frame from the sprite map, if the sprite map in question contains multiple frames. The final fourth argument can be used to turn trilinear filtering on or off for this sprite.

See also **@MAP**, **@FRAME**, and **@FILTER**.

SPRITEMAP aexpr,sexpr[,aexpr[,aexpr]]

This command loads the image file specified in the second argument, and if the file can be successfully decoded, it will be stored in the sprite map list under the numerical ID specified in the first argument. The optional third and fourth arguments are interpreted as the number of frames stored in the sprite map per row, and per column. They both default to one. If the path to the source file is relative, 8ASIC will search for it in the documents folder, and in the demo folder.

SPRITEPOS aexpr,aexpr,aexpr[,aexpr[,aexpr[,aexpr]]]

Sets the transformation of the sprite with the number specified in the first argument. The second and third arguments are interpreted as X and Y position. The optional fourth argument is the rotation in degrees or radians, depending on the current angular unit. The optional fifth argument, if it's the last one, sets proportional scale. If a sixth argument is also provided, the fifth and sixth arguments are interpreted as separate X and Y scale.

The origin point for these transformations is the sprite's center.

See also **@POSX**, **@POSY**, **@ROTATION**, **@SCALE**, **@SCALEX**, and **@SCALEY**.

STOP

Stops the execution of the program, and prints "**STOPPED AT LINE**" followed by the line number where it was stopped. Execution can be resumed using the **CONT** command.

TAB aexpr[,aexpr]

Moves the cursor to the specified column of the current line, or the next line if the target column is to the left of the current cursor position. If two arguments are provided, the command works like **AT** instead. See also **TAB()**.

TEXT aexpr,aexpr,expr[;expr][,expr][...]

A combination of **AT** and **PRINT** in one command. The first two arguments are interpreted as the coordinates for the **AT** or **POSITION** command, and the results of the remaining expressions are printed at that position.

TIME

Seconds elapsed since 8ASIC started. This value is precise to a couple of milliseconds, and can be used for animation timing. See also **@TIME$**.

TRACER bexpr
TURTLE bexpr

Sets the opacity of the built-in turtle sprite from 0 (fully transparent, default) to 1 (fully opaque). Note that turtle graphics is available regardless of whether the turtle sprite is visible or not.

TRAP aexpr

TRAP can be used to set a line number where 8ASIC jumps when it encounters an error, instead of printing an error message and stopping. The error message will still be stored in **@ERROR$**, and the line number, where the error occurred, in **@ERRORLINE**. Note that after an error has been “trapped” in this fashion, the trap line resets, and needs to be set again.

VECTOR

Sets the current graphics layer (background or foreground) into vector mode, which means that any graphics objects drawn onto this layer will preserved as vectors, and rasterized every time the screen is refreshed. This allows you to draw a picture that's much larger than the screen, and show different parts of it by changing the **@NUDGE** system variable. See also **RASTER**.

VERTEX aexpr,aexpr[,expr[,expr]]
VERTEXR aexpr,aexpr[,expr[,expr]]

Adds a new vertex to the current polygon. If no polygon is currently active, **VERTEX** starts drawing a new one. The coordinates passed to **VERTEXR** are interpreted as relative. New polygons (like lines) can also be started explicitly using the **PLOT** or **MOVE** commands. 8ASIC supports polygons with an arbitrary number of points. Note that even though adding points to a polygon is quite fast, it takes progressively longer as their number increases, since the polygon has to be converted to triangles (tessellated) every time.

The third and fourth arguments, if provided, are interpreted as pen and brush colors. If they are strings, these will be looked up in the palette of named colors.

See also **DRAW**, **DRAWR**, **MOVE**, **MOVER**, **PLOT**, and **PLOTR**.

VOICE expr

Choose the voice that will be used for **SAY** commands. The argument can either be a one-based index (up to **@VOICES**) or a string containing the voice name.

Built-in Functions

ABS(aexpr)

Absolute value.

ALPHA(aexpr)

Returns the alpha (opacity) value of a color.

ASC(sexpr)
CODE(sexpr)

UTF32 code of the first character of the string argument.

ATN(aexpr)

Arc tangent. See also **DEG** and **RAD**.

ATN2(aexpr,aexpr)

Arc tangent taking two arguments, Y and X. Returns a correct result for all 4 quadrants of the circle. See also **DEG** and **RAD**.

BLEND(aexpr,aexpr,aexpr)

Interprets the first two arguments as colors, and blends them using the third argument as the blending factor. The alpha value of the second color is multiplied with the blending factor, then the second color is blended over the first.

BLUE(aexpr)

Returns the blue component of a color.

CHR$(aexpr)

Character corresponding to UTF32 code in the argument.

CLAMP(aexpr,aexpr,aexpr)

Returns the first argument clamped to the range formed by the second and third arguments.

CLOG(aexpr)

Base 10 logarithm.

COLOR(expr)

If the expression in the argument is a string, it is interpreted as the name of a color to be retrieved from the palette. If a color with such a name doesn't exist, **COLOR()** will return zero.

If the input expression is numerical, it is interpreted as a one-based index into the alphabetically-sorted palette. If the index is negative, it is also interpreted as an index into the palette, but sorted by hue and luminance instead. The total number of colors in the palette can be queried using **@COLORS**.

COLOR$(aexpr)

Returns the name of the color in the argument, or an empty string if that color is not in the palette.

COS(aexpr)

Cosine. See also **DEG** and **RAD**.

DEC(sexpr)

Convert a hexadecimal number in a string to a number.

EXP(aexpr)

The number e raised to the power specified in the argument.

FRAC(aexpr)

Returns the fractional part of the value passed in the argument.

GREEN(aexpr)

Returns the green component of a color.

HEX$(aexpr)

Returns the hexadecimal representation of the number in the argument.

INT(aexpr)

Returns an integer that is less than or equal to the argument.

INSTR(sexpr,sexpr[,aexpr])
UINSTR(sexpr,sexpr[,aexpr])

Returns the 1-based position of the substring (second argument) within a string (first argument) or 0 if not found. The optional third argument can be used to start the search from a position other than 1. The **UINSTR** variant is case-insensitive.

LCASE$(sexpr)

Returns the string in the argument converted to lowercase.

LEFT$(sexpr,aexpr)

Returns the specified number of characters from the start of a string.

LEN(sexpr)
LEN(array[,aexpr])

Returns the length of the string expression or array in the argument. For multi-dimensional arrays, the size of the last dimension is returned, unless a dimension is specified in the second argument.

LOG(aexpr)

Returns the natural logarithm of the input expression.

MAX(aexpr,aexpr)

Returns the larger of the two arguments.

MID$(sexpr,aexpr,aexpr)

Returns the specified number of characters (third argument) from the middle of the string (second argument). The same functionality can be achieved by the substring syntax, so this function is included only for compatibility.

MIN(aexpr,aexpr)

Returns the smaller of the two arguments.

MIX(aexpr,aexpr,aexpr)

Mixes two colors (first two arguments) together, using the third argument as a blending factor.

RIGHT$(sexpr,aexpr)

Returns the specified number of characters from the end of a string.

RND(aexpr)

Returns a random non-negative real number lesser than the value of the input argument. If the argument is 0, it's treated as if it was 1 (for compatibility reasons). If the argument is negative, the random number generator will be re-seeded from the system time (like on the C64).

RAND(aexpr)

Returns a random non-negative integer lesser than the value passed in the argument.

RED(aexpr)

Returns the red component of a color.

RGB(aexpr,aexpr,aexpr)

Combine red, green, and blue components (0...255 range) into a color. The alpha component of the resulting color will be set to 255 ($FF).

RGBA(aexpr,aexpr,aexpr,aexpr)

Combine red, green, blue, and alpha components (0...255 range) into a color.

ROUND(aexpr)

Round to the nearest integer.

SGN(aexpr)

Returns -1 if the value of the argument is negative, 1 if positive.

SIN(aexpr)

Sine. See also **DEG** and **RAD**.

SQR(aexpr)

Square root.

STR$(aexpr[,aexpr])

Converts the number in the first argument to a string. The second, optional argument controls the precision (number of significant digits) of the conversion.

TAB(aexpr,[aeprx])

This function can only be used within **PRINT** statements. It moves the cursor to the specified column of the current line, or the next line if the target column is to the left of the current cursor position. If two arguments are provided, the function works like the **AT** command instead. **TAB()** also acts as an expression separator, similar to a semicolon or a comma. See also **TAB** (keyword).

TAN(aexpr)

Tangens. See also **DEG** and **RAD**.

TRUNC(aexpr)

Round towards zero.

UCASE$(sexpr)

Returns the string in the argument converted to uppercase.

VAL(sexpr)

Converts a number in a string to a normal number.

Operators

This table lists all operators in the order of decreasing precedence. Operators with equal precedence are grouped together.

Operator	Result
+ (with strings)	Concatenated strings
= (with strings)	True (1) if strings are equal
<> (with strings)	True (1) if strings are not equal
<= (with strings)	True (1) if the first string is lesser or equal
>= (with strings)	True (1) if the first string is greater or equal
< (with strings)	True (1) if the first string is lesser
> (with strings)	True (1) if the first string is greater
- (unary)	Negative value
+ (unary)	Unchanged, operator is purely cosmetic
NOT (unary)	True (1) if operand is false (0)
^ or ******	First number raised to the power of the second
SHR	First integer bit-shifted right by the second
SHL	First integer bit-shifted left by the second
&	Bitwise AND of the two integers
!	Bitwise OR of the two integers
%	Bitwise XOR (exclusive or) of the two integers
*****	Multiplication of the two operands
/	Division of the first operand by the second operand
MOD	The remainder of the division of the first by the second operand
DIV	Integer division of the first operand by the second operand
+	The sum of the two operands
-	The difference between the first and second operands
= (with numbers)	True (1) if the operands are equal
<> (with numbers)	True (1) if the operands are not equal
<= (with numbers)	True (1) if the first operand is lesser or equal
>= (with numbers)	True (1) if the first operand is greater or equal
< (with numbers)	True (1) if the first operand is lesser
> (with numbers)	True (1) if the first operand is greater

System Variables

Instead of **PEEK** and **POKE**, 8ASIC has a set of special system variables, which control various aspects of the virtual machine. To avoid polluting the global namespace with many new keywords, all system variable names start with an @ sign. They can be simple scalar variables or arrays, and some can even be both.

@ATTACK

This 16-element array controls the attack times (in seconds) of the ADSR envelopes of the individual software synthesizer voices. Attack is the time it takes for the sound to reach peak volume after "note on". See also **@DECAY**, **@SUSTAIN**, and **@RELEASE**.

@BLENDMODE

This system array controls the blending modes of individual sprites. A value of 1 means normal blending, 2 is "screen" mode blending. More blending modes may be added in the future.

@BRUSH

If not subscripted, **@BRUSH** controls the current brush color. The brush color is used to draw text character backgrounds, and to fill polygons drawn using the **VERTEX** command. If subscripted, (i.e. **@BRUSH(X, Y)**) it controls the background color of the text character at the coordinates [X,Y], which are zero-based, with the origin in the top left corner.

@BRUSHDOWN

Controls whether the turtle's brush is down (painting) or up (not painting). See also **BRUSHDOWN**.

@BUTTON

This read-only array contains the current state of all mouse buttons (up to 32). The left button is at index 1, the right button at index 2, the middle button at index 3, and any additional buttons at the remaining indices. If an item has the value of 1, it means the corresponding button is currently pressed. See also **@CLICK**, **@DBLCLICK**, **@MOUSEX/Y** and **@WHEEL**.

@CHAR

If not subscripted, **@CHAR** controls the character code at the location of the text cursor. If subscripted, (i.e. **@CHAR(X,Y)**) it can be used to read or set the character code of any screen character.

@CLICK

This array contains the positions (in graphics coordinates) where each of the mouse buttons was last pressed. The first subscript selects the button, (1 to 32) and the second one the X or Y coordinate (1 or 2). By setting entries in the array to impossible values, (like -1000) and then waiting until they change back to valid positions, you can track whether mouse buttons were clicked. See also **@BUTTON**, **@DLBCLICK**, **@MOUSEX/Y**, and **@WHEEL**.

@CLIP

The amount of clipping applied to the waveform of each software synthesizer voice. Zero means no clipping (which produces a saw, triangle, or sine waveform, depending on other parameters), while one means full clipping (square wave-form). Default is one, full clipping. See also **@NOISE**, **@PULSE**, **@SINE**, and **@SLOPE**.

@COLORS

This read-only variable holds the total number of named colors in the 8ASIC col-or palette.

@COLUMNS
@LINES

These two system variables control the number of text columns and lines dis-played of the screen. **@COLUMNS** sets the number of columns, and **@LINES** the number of lines. Note that these values are not independent from each other, but setting one changes the other, depending on the size and aspect ratio of the 8ASIC window. Changing the text resolution will also change the range of graph-ics coordinates.

@CURSOR

@CURSOR controls whether the text cursor is visible (1) or not (0). See also **CURSOR**.

@CURSORX
@CURSORY

These two system variables together control the location of the text cursor. **@CURSORX** controls the X coordinate, and **@CURSORY** the Y coordinate. See also **AT**.

@DATE$

Contains the current date as a string.

@DBLCLICK

This array contains the positions (in graphics coordinates) where each of the mouse buttons was last double-clicked. The first subscript selects the button, (1 to 32) and the second one the X or Y coordinate (1 or 2). See also **@BUTTON**, **@CLICK**, **@MOUSEX/Y**, and **@WHEEL**.

@DECAY

This array controls the decay times (in seconds) of the ADSR envelopes of the individual software synthesizer voices. Decay is the time it takes for the sound to reach the sustain volume after the attack. See also **@ATTACK**, **@SUSTAIN**, and **@RELEASE**.

@ERROR$

Contains the last error message.

@ERRORLINE

Contains the line number of the last error, or -1 if the error occurred in direct mode.

@FILTER

This system array controls whether individual sprites are filtered (using trilinear interpolation) or not (nearest neighbor "pixelated" interpolation).

@FRAME

This system array controls which frame is being shown by each sprite. Of course this works only if the sprite map associated with a sprite has multiple frames. The default is to show the first frame.

@GRAPHX
@GRAPHY

These two system variables control the origin, and the range of coordinates for graphics commands (**PLOT**, **DRAWTO** and **VERTEX**). **@GRAPHX** sets the X (horizontal) origin and range, while **@GRAPHY** sets the Y (vertical) origin and range. The default value for both is the text screen size times 8.

The default origin (zero) of graphical coordinates is in the bottom left corner of the screen. To move the X or Y origin to the opposite edge of the screen, set one or both elements of the array to a negative value. **@GRAPHY=-200** sets the range of Y coordinates to 200, and the origin to the top edge of the screen.

See also **GRAPHICS**.

@HEADING

Controls the turtle's absolute heading. See also **HEADING**.

@INTERRUPTS

This system variable controls whether interrupts are enabled. Note that interrupts work only while the program is running, regardless of the value of this variable. See also **INTERRUPTS**.

@KEY

This read-only array allows you to find out whether a particular key is being held down. The key codes to be used as indices into this array can be obtained from the **GET** command. The array holds ones for all keys which are currently being held down, and zeroes for all others.

@KEYVOLUME

This system variable controls the volume of the sound played after each keystroke. The range of values is 0 (key sound off) to 1 (full volume), with 0.5 being the default. The value of this system variable will be saved to the settings file, and restored next time 8ASIC is started.

@LAYER

This system array controls the screen layer (drawing order) of each sprite. Valid values are from 1 (behind all layers) to 4 (in front of all layers).

@LFO

The frequency (in Hz) of the low-frequency oscillator associated with each synthesizer voice. The low-frequency oscillator drives the tremolo and vibrato effects. The default frequency is 0, meaning the LFO is disabled.

@LOWPASS

The frequency (in Hz) of the low-pass filter applied to each synthesizer voice. The default value is 10 Khz, and the range is from 50 Hz to 20 Khz.

@MAP

This system array controls which sprite map is associated with each sprite. The default value of 0 means “none”, and has to be changed to a valid ID of a loaded sprite map, in order for the sprite to appear on screen.

@MARGINL
@MARGINR

These two system variables control the left and right margins, used when printing text on the screen. **@MARGNL** controls the left margin, and **@MARGINR** the right margin. Default values are 2 left and 0 right.

@MOUSEX
@MOUSEY

These two system variables contain the current mouse position in graphics coordinates. By changing one or both coordinates to an impossible value (like -1000), you can track whether the mouse moved. See also **@BUTTON**, **@CLICK**, **@DBLCLICK**, and **@WHEEL**.

@NOISE

The amount of noise added to the waveform of each software synthesizer voice. Zero means no noise, while one completely replaces the waveform with noise. The noise has the same frequency as the sound. Default is zero, no noise. See also **@CLIP**, **@PULSE**, **@SINE**, and **@SLOPE**.

@NUDGE

Shifts screen layers (background, text, foreground) by the specified distance in graphics coordinates. The first subscript is the layer to be shifted, with 1 for the background, 2 for text, and 3 for the foreground. The second subscript is the axis, with 1 for X and 2 for Y. The nudge distance is not limited in any way – you can just slightly shift a layer or move it completely off the screen. See also **@OVER-SCAN**.

@OPACITY

This system array controls the opacity of each particular sprite. Valid values are from 0 (fully transparent) to 1 (fully opaque).

@OVERSCAN

If this Boolean system variable is set to a non-zero value, it adds two characters worth of overscan to all edges of the screen. This ensures that at least one row or column of characters is fully off the screen on each of the four sides. Overscan in conjunction with **@NUDGE** allows you to smoothly scroll the text screen, because you can hide the addition of new lines or columns behind the edge.

@PAN

This array controls the stereo positions of the individual software synthesizer voices. A value of -1 pans the respective voice hard left, 1 hard right, and 0 (default) is the center. The left & right volume levels are calculated using the -3 dB rule. See also **@VOLUME**.

@PAPER

This system variable controls the overall background color of the screen, which shows through wherever there's nothing drawn or written.

@PEN

If not subscripted, **@PEN** controls the current pen color. The pen color is used to draw text characters, points, lines, and to outline polygons drawn using the **VERTEX** command. If subscripted, (i.e. **@PEN(X,Y)**) it controls the color of the text character at the coordinates X and Y.

@PENDOWN

Controls whether the turtle's pen is down (drawing) or up (not drawing). See also **PENDOWN**.

@PITCH

This array controls the relative pitch (in octaves) of the individual software synthesizer voices. A value of 0 (default) means the respective voice generates the exact pitch passed to the **SOUND** command. Any other value will be added to the pitch passed to **SOUND** to calculate the effective pitch.

@PLOTX
@PLOTY

These two system variables contain the last X and Y coordinates given to any graphics command (**MOVE**, **PLOT**, **DRAWTO**, **VERTEX**, **BOX** or **CIRCLE**). They are also updated when moving the turtle using **FORWAD** or **BACKWARD**. Setting them is equivalent to the **MOVE** command.

@PORTAMENTO

This system array allows the pitch of a synthesizer voice to vary slowly over time, instead of changing instantly. The value in the array is the time in seconds it will take for the voice to reach a new pitch. This works only if a voice is already playing a sound, and a new **SOUND** command with a different pitch is issued for that voice. To change pitch instantly with a non-zero **@PORTAMENTO** value set, trigger a “note off” using **SOUND** with just the voice number argument, then play the second sound immediately after that.

@POSX
@POSY

These two system arrays control the horizontal and vertical position of sprites (in graphics coordinates).

@PULSE

The pulse width of the waveform of each software synthesizer voice. In order for this parameter to have any effect, the waveform needs to be clipped to some extent, and the effect of pulse width increases with clipping. Default is 0.5, a neutral position. See also **@CLIP**, **@NOISE**, **@SINE**, and **@SLOPE**.

@RELEASE

This array controls the release times (in seconds) of the ADSR envelopes of the individual software synthesizer voices. Release is the time it takes for the sound to go down to zero volume after "note off". See also **@ATTACK**, **@DECAY**, and **@SUSTAIN**.

@ROTATION

This system array controls the rotation of sprites (default is 0).

@SCALE
@SCALEX
@SCALEY

These three system arrays control the proportional, horizontal and vertical scale of sprites (all defaulting to 1).

@SINE

This synthesizer voice parameter allows you to trasnform the normally straight "slopes" of saw and triangle waveforms into more or less smooth S-shaped transitions, based on the sine function. This renders the resulting sound less harsh. Note that in order for this parameter to have any effect, the clipping level of the respective channel needs to be lower than one. The defaut is 0, or no amount of sine smoothing in the slopes. See also **@CLIP**, **@NOISE**, **@PULSE**, and **@SLOPE**.

@SIZEX
@SIZEY

These two system variables hold the last width and height given to a **BOX** command, or the last X and Y radius given to a **CIRCLE** command. You can set these values, but it won't affect the box or circle that was already drawn on the screen.

@SLOPE

The overall left/right slope of the waveform of each software synthesizer voice. In order for this parameter to have any effect, clipping needs to be less than one. If that is the case, a slope of 0 produces a left-leaning saw waveform, a slope of 1 a right-leaning saw waveform, and a slope of 0.5 a symmetrical triangle waveform. A slope of 0.5, with 0 clipping and 1 sine produces a pure sine waveform. Default slope is 1, right-leaning saw waveform. See also **@CLIP**, **@NOISE**, **@PULSE**, and **@SINE**.

@SUSTAIN

This array controls the (linear) sustain volumes of the ADSR envelopes of the individual software synthesizer voices. After the initial attack and decay, the sound reaches the sustain volume, and stays there until "note off". See also **@ATTACK**, **@DECAY**, and **@RELEASE**.

@TEXTLAYER

Controls where in the drawing order the text screen will be drawn. The default value of 2 means between the background and foreground graphics layers. A value of 1 moves the text screen behind the background, while a value of 3 moves it in front of the foreground.

@TIME$

Contains the current time as a string (read-only).

@TREMOLO

This system array controls the amount of LFO-based amplitude (volume) variation for each of the synthesizer channels. Valid values are from 0 (no variation) to 1 (maximum variation).

@TURTLE

Sets the opacity of the built-in turtle sprite.

@VIBRATO

This system array controls the amount of LFO-based frequency variation for each of the synthesizer channels. Valid values are from 0 (no variation) to 1 (+/- one octave).

@VOICE

Controls which voice will be used when **SAY** commands are issued. Valid values are from 1 to **@VOICES**.

@VOICES

Contains the total number of text-to-speech voices available to 8ASIC. On Windows, the situation is somewhat complicated, because there are two text-to-speech APIs, and 8ASIC can use only some of the voices right now. On the Mac, this problem doesn't exist, and my Macbook has no less than 47 different voices to choose from. Keep in mind that changing the voice usually also changes the language. See also **VOICE**, **@VOICE**, **@VOICENAME$**, and **@VOICE-LOCALE$**.

@VOICENAME$

A read-only string array containing the names of the voices corresponding to **@VOICE** values.

@VOICELOCALE$

A read-only string array containing the locales of the voices corresponding to **@VOICE** values. Locales are in the format **"en-US"**, **"en-GB"**, **"de-DE"**, **"it-IT"** and so on.

@VOLUME

This array controls the relative (linear) volumes of the individual software synthesizer voices. A value of 0 effectively turns the voice off, while 1 (default) means full volume. Note that the final volume of sound played through each voice is also affected by the voice's ADSR envelope, and the volume argument passed to the **SOUND** command. See also **@PAN**.

@WHEEL

Contains the mouse's wheel position. The system variable can be reset to zero, or any other value, to better track the relative wheel rotation. See also **@BUTTON**, **@CLICK**, **@DLBCLICK**, and **@MOUSEX/Y**.

@WINDOWX
@WINDOWY

These two read-only system variables hold the X and Y size of the client area of the 8ASIC window in device pixels.

Special Characters

8ASIC's font contains many special characters from the ATASCII and PETSCII character sets. The following table lists they key combinations that can be used to type these characters, as well as their codes.

Key	Character w/ Alt	Character w/ Alt+Shift	UNICODE	UNICODE w/ Shift
1	⚀	◢	U+2680	U+E000**
2	⚁	◣	U+2681	U+E001**
3	⚂	◥	U+2682	U+E002**
4	⚃	◤	U+2683	U+E003**
5	⚄		U+2684	
6	⚅		U+2685	
A	├	◖	U+251C	U+E004**
B	▕		U+2595	
C	┘	╯	U+2518	U+256F
D	┤	◗	U+2524	U+E005**
E	┐	╮	U+2510	U+256E
F	╱	▞	U+2571	U+259E
G	╲	▚	U+2572	U+259A
H	◢	◤	U+25E2	U+25E4
I	▗	▛	U+2597	U+259B
J	◣	◥	U+25E3	U+25E5
K	▝	▙	U+259D	U+2599
L	▘	▟	U+2598	U+259F
M	▔	♛	U+2594	U+265B
N	▁	♚	U+2581	U+265A
O	▖	▜	U+2596	U+259C
P	♣		U+2663	
Q	┌	╭	U+250C	U+256D
R	─		U+2500	
S	┼	╳	U+253C	U+2573
T	●	◯	U+25CF	U+25EF
U	▄	▀	U+2584	U+2580
V	▏		U+258F	
W	┬	◓	U+252C	U+E006**
X	┴	◒	U+2534	U+E007**
Y	▌	▐	U+258C	U+2590
Z	└	╰	U+2514	U+2570
;	♠	♟	U+2660	U+265F
\	│	♞	U+2502	U+265E
,	♥	♜	U+2665	U+265C
.	♦	♝	U+2666	U+265D
Left	←	◀	U+2190	U+25C0
Right	→	▶	U+2192	U+25B6
Up	↑	▲	U+2191	U+25B2
Down	↓	▼	U+2193	U+25BC
Space*	█	▒	U+2588	U+2592

* To circumvent the global Alt+Space hotkey in Windows, press Ctrl+Alt+Space instead.

** UNICODE character codes starting from U+E000 are reserved for custom characters, and the glyphs assigned to them are specific to 8ASIC. They won't be visible if you open your program's source text in an external editor.

Included Demos

3D Plot

3D graphs of functions plotted using the floating horizon algorithm. Shows how you can redefine functions using **DEF FN**.

Babble

Says random gibberish sentences using random text-to-speech voices. Demonstrates text-to-speech.

Boom

Shows how to synthesize simple sound effects, and how to shake the screen layers using the **@NUDGE** system array.

Bounce

Draws a colorful triangle between 3 points bouncing around the screen. Shows per-vertex fill colors and the resulting gradients.

Bubbles

Draws rising bubbles. Shows the **CIRCLE** function, and transparency.

Color Quiz

Tests your knowledge of obscure color names. Demonstrates the use of the palette of named colors, and basic keyboard input.

Dice

Shows how to create simple graphics (frames) using special characters.

Eliza

A talking version of the classic program Eliza.

Game of Life

Conway's game of life, using special characters to represent the various cell states.

Hello World

The classic.

Interference

Demonstrates how to use **VECTOR** mode to draw graphics that are larger than the screen.

Koch

Draws the Koch snowflake using turtle graphics.

Lines

Draws a bunch of random lines on the screen. Shows random pen colors per vertex, and the resulting color gradients.

Mandelbrot

Draws the Mandelbrot set by using the background color of text characters as large pixels.

Matrix

Simulates the notorious “raining code” effect from the Matrix movie.

Maze

A classic minimalist BASIC program. Shows some special character usage.

Monster

A small sprite demo, using one of my daughter's drawings as a sprite. Apart from sprites, it also shows how to create simple sound effects.

Palette

Utility program to show all built-in colors, either sorted by name or by hue. Demonstrates slightly more advanced keyboard input, and the palette of named colors.

Raytracer

An enahnced port of a 2008 Commodore 64 BASIC demo by Marco64. It shows how to render coarse pixel graphics using the BOX command.

Queens

Calculates all solutions for the “8 Queens” problem, using simulated recursion. Also shows some special character usage. For real recursion with local variables, see the Tree demo.

Quilt

A variation of the Maze demo, but with random colors. Shows how bitwise operations can be used to create color contrast.

Spirals

Draws spirals based on arbitrary input shapes. Shows how background and foreground layers can be used to draw independently moving graphics.

Smooth Maze

Another version of Maze, but with overscan and smooth scrolling, as described here.

Synth

A graphical front-end for 8ASIC's built-in software synthesizer, demonstrating all of its functions.

Tree

Draws a tree using a recursive L-system algorithm and turtle graphics. Demonstrates the use of local variables.

Open Source Licenses

8ASIC uses the **Qt** framework, licensed under the LGPL v3.

It also uses the **FreeType** library, licensed under the FreeType Project License.